Integrating Computer Log Files for Process Mining: a Genetic Algorithm Inspired Technique

Jan Claes, Geert Poels

Department of Management Information Systems and Operations Management
Faculty of Economics and Business Administration
Ghent University, Tweekerkenstraat 2, 9000 Ghent, Belgium
{jan.claes,geert.poels}@ugent.be

Abstract. Process mining techniques are applied to single computer log files. But many processes are supported by different software tools and are by consequence recorded into multiple log files. Therefore it would be interesting to find a way to automatically combine such a set of log files for one process. In this paper we describe a technique for merging log files based on a genetic algorithm. We show with a generated test case that this technique works and we give an extended overview of which research is needed to optimise and validate this technique.

Keywords: Tool-Support for Modeling, Business Process Modeling, Process Mining, Process Discovery, Log File Merging

1 Introduction

When people are discussing organisational processes in a certain context they mostly want to find out how processes *should be*. A good starting point for such discussions is to discover processes as they *currently are*. This can be done by asking domain experts to model the organisation's current processes as they know or perceive them. This manual activity is, however, time-consuming [1] and subjective [1, 2].

Process discovery[1] techniques analyse actual process data, which can be found in computer log files, to automatically discover the underlying process. The main idea in process discovery techniques is that business processes are executed many times and thus log files contain many traces of the same process [2]. It is assumed that each trace contains the different events registered for one process execution in the correct order [3]. By searching for repeating patterns in these traces a general process flow can be discovered. An overview of process discovery and other process mining techniques can be found in [1-5].

[1] The IEEE Task Force on Process Mining, consisting of major researchers and practitioners in the Process Mining field, decided to reserve the term *process mining* as an umbrella term for all techniques on process data and to use the term *process discovery* for techniques to reveal processes out of process data. (http://www.win.tue.nl/ieeetfpm/)

Process mining techniques are developed to be applied to single computer log files. But many business processes are supported by different software tools and their executions are by consequence recorded into multiple log files. Consider for example a back-end IT support process where support actions are logged through a ticketing system and changes to program code and documentation are logged using a version control system. Part of the process is registered in a first log file and the other part is registered in a second log file. We are not aware of any *automated* way to use both files at once for process mining in general and process discovery in particular.

Therefore, it would be interesting to find a way to automatically combine such a set of log files before process discovery techniques are applied. In this paper we describe a technique for merging log files containing different traces of the same process executions based on a genetic algorithm. Genetic algorithms are search heuristics inspired by natural selection and the principle of survival of the fittest [6]. We implemented our log file merging algorithm in the latest version of ProM [3], a well-known academic process mining tool, and used simulated test data to demonstrate its effectiveness for process discovery.

We start with a description of process discovery and its assumptions in Section 2, followed by a description of the research problem and the assumptions we make for its solution. Our genetic algorithm inspired technique for merging log files is presented in Section 3. The proof of concept application using a generated test case is presented in Section 4. To end the paper, an extended overview is given of which further research is needed to optimise and validate our solution in Section 5 and a conclusion is provided in Section 6.

2 Problem Description

2.1 Process Discovery

Process discovery is based on a number of assumptions [2]. First, process data needs to be available somehow, somewhere. Second, it should be possible to distinguish data from different executions and group the logged event data in traces (one trace per execution). Similarly, it has to be possible to distinguish events and therefore it is assumed that different events have different names [2]. However, an approach to discover processes with duplicate tasks, where different logged events have the same name, can be found in [7]. Third, the order of logged events should be clear [3], but it is not always strictly necessary to have exact timestamps (e.g. the Little Thumb technique does not use timestamps [8]). Of course, some process discovery techniques make more assumptions like *all data should be logged (completeness)* or *there should be no errors or noise in the logged data (correctness)* [5].

Given these assumptions it is possible to compose log files consisting of a group of traces which themselves consist of groups of ordered events. This is the starting point for process mining techniques [2]. In this paper, we focus mainly on process discovery techniques. These techniques look for repeating patterns in the traces of the log file. If an event A is immediately followed by an event B in every trace of the log,

it can be assumed that in the underlying process task B always has to follow task A. This results in drawing an edge between A and B in the model of the discovered process. If A is in a number of traces immediately followed by B but in all other traces immediately preceded by B, it can be assumed that A and B are in parallel (i.e. and-split). If A is in a number of traces immediately followed by B (but not by C) and in all other traces immediately followed by C (but not by B), it can be assumed that after A comes B or C (i.e. or-split). This simplified example describes the basics of the α-algorithm [9], which is the basic theoretical algorithm for process discovery.

2.2 Merging Log Files

The availability of data on process executions is a logical assumption for automatic process discovery. Furthermore, it is also clear that if parts of the process are handled manually and these manual activities are not registered, it is almost impossible to recover these steps in the process with process discovery techniques. In some cases one could guess a hidden task in the discovered process [5]. But what if the process to be discovered is supported by multiple computer-based tools or systems and therefore parts of the process are recorded in another log file?

Consider for example the back-end IT support process in Fig. 1. Two types of support requests could reach the back-end unit: requests for change (RFC) and requests for information (RFI). All white tasks are supported by a ticketing system (e.g. Omnitracker) and all gray tasks can be discovered in the logging of a version control system (e.g. Subversion).

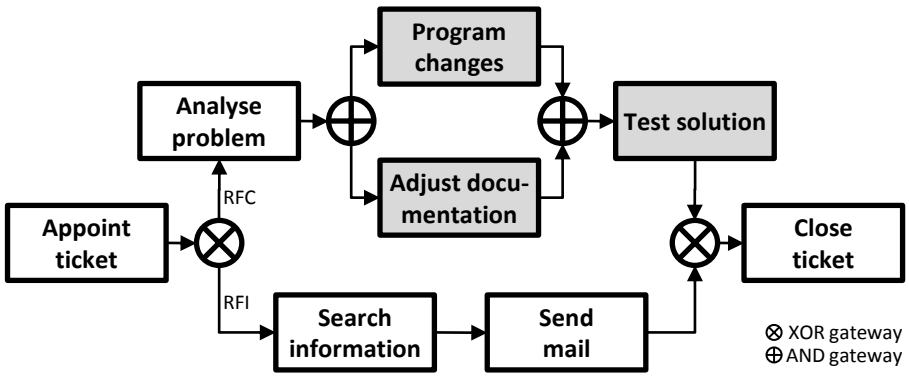


Fig. 1. Example process model for a back-end IT support process

All current process discovery (or even process mining) techniques start from a single event log file containing traces of ordered events that register the process executions. If execution traces are split up in different log files, then existing process discovery techniques can still be used provided that the different log files containing partial evidence of the process executions are first merged into one more complete log file.

We focus our solution on tackling the problem of one process whose executions are being logged in two log files, abstracting from more complex problems that are variations to this basic problem (e.g. more than two log files, more than one process). In other words, both log files contain partial data about executions of the same process. For the example process of Fig. 1, the execution of the white tasks is registered in one log and the execution of the gray tasks is registered in the other log. Our solution firstly links together the traces of both logs that belong to the same process execution and secondly merges the events of linked traces from both log files into one trace in the merged log.

In the solution proposed in this paper we make two additional assumptions. First, for the linking of traces in both logs we assume that the second log contains no process start events. This means that all traces from the second log are initiated in the first log. In our opinion, this assumption makes sense if processes have only one starting point (that is always supported by the same software and is thus logged in the same file). The linking problem is then reduced to linking each trace from the second log to only one trace from the first log containing the start event of the same process execution. This assumption includes that we know which of both logs is the first (containing all process start events). In the example shown in Fig. 1, the log maintained by the ticketing system contains the registered start events of the back-end IT support process executions. Hence, the problem reduces to linking each trace in the log of the version control system to one trace in the log of the ticketing system.

Second, for the merging of the events from the linked traces we assume that reliable and comparable timestamps for all events are available (we consider missing or incorrect time information to be noise).

- Timestamps are *reliable* if the events are logged when they occur (e.g. if a worker registers a phone call two hours late, the timestamp will probably not be reliable) and if they are logged correctly (e.g. not the case if the software itself has errors or if the operating system provides inaccurate time information to the software).
- Timestamps are *comparable* if their precision is (almost) equal (this is not the case if in one log file only dates were recorded and the other has millisecond precise time information) and if they are in the same time zone (or can be converted to the same time zone).

If these two conditions are met, we can simply order events from linked traces in chronological order. This makes sense because, given our assumptions, this is the order in which the events actually occurred. Hence, in our solution description we focus on the linking problem to find out which traces in both logs belong together.

3 Solution Design

Many factors can indicate whether traces from two logs relate to the same process execution (see 3.2.2). Instead of focussing on one such factor we incorporated multiple factors in our solution. The more elements indicate that two traces should be linked, the more certain we are that they in fact relate to the same process execution.

To design our solution we looked for similar problems in other domains and found our inspiration in the principles of genetic algorithms.

3.1 Genetic Algorithms

Genetic algorithms are inspired by nature and rely on Darwin's principle of "survival of the fittest" [10]. A population of multiple solutions to a certain problem evolves from generation to generation with the goal to be optimised through generations. A genetic algorithm starts with a totally random initial population. Each next generation is constructed with bits and pieces of the previous generation using crossover and mutation operators. Crossover means that random parts of two solutions are swapped and mutations are random changes in a solution. Mutation is needed to assure that lost parts can be recovered. To quantify the quality of solutions in a generation, a fitness function is used. Because the fitness score determines the chance to be selected as input for a next generation of solutions, consecutive generations tend to get better in every step (i.e. survival of the fittest). The algorithm ends when a termination condition is met. This is mostly when a certain measurable degree of correctness is achieved or when no more optimisation is established. The steps of a genetic algorithm can be summarised as: (i) initialisation, followed by a repeating sequence of (ii) selection based on the fitness function, (iii) reproduction using crossover and mutation, and ended by (iv) a termination condition.

For more information about genetic algorithms we refer to [6].

3.2 Genetic Algorithm Inspired Technique for Merging Computer Log Files

We start with a random combination of links between the traces contained in both logs. Then in each step we make random mutations by changing the links. We calculate the fitness score before and after all mutations of each step. We continue with the original version if the fitness diminished, otherwise we continue with the new version.

To compare with genetic algorithms, our solution can be regarded as a genetic algorithm with a solution population of size one. Only one combination of links (i.e. one solution) is used as a base for finding an optimal solution. Selection for reproduction and crossover are useless if there is only one element in the population. We end up with only initialisation, mutations evaluated by a fitness function and termination. The same simplified version of genetic algorithms is used for timetabling in Genetic Algorithm for Time Tabling (GATT) [11]. The expert reader will notice that because of the simplifications this technique is no real genetic algorithm anymore and for that reason we call it a *genetic algorithm inspired technique*.

3.2.1 Initialisation

Because we assume each second trace (each trace in the second log) was initiated in the first log, we have to link each second trace to only one first trace (trace in the first

log). We have to loop over all second traces and assign each of them to a random first trace. We do not assume that each first trace can initiate only one second trace which means that multiple second traces can be linked to the same first trace (see Fig. 2).

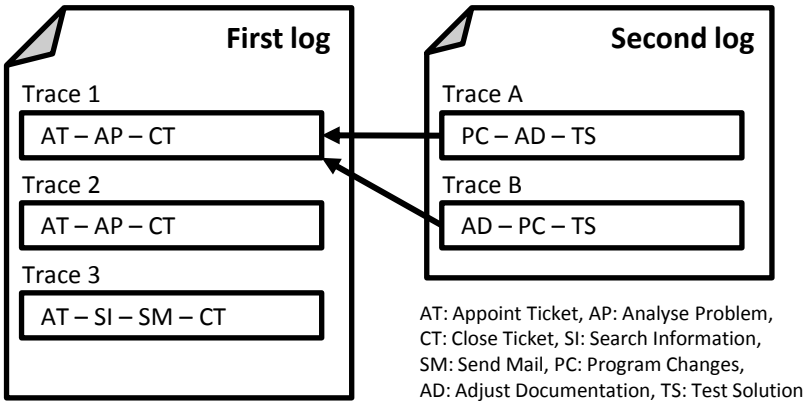


Fig. 2. Linking second traces (traces from the second log) to first traces (from the first log)

3.2.2 The Fitness Function

In each step we compare the fitness score before and after mutations to decide which version to continue with. As a result of not knowing the actual relation between the two log files we start from working hypotheses describing specific conditions that could indicate that two traces should be connected. We treat these different factors indicating traces from the two logs should be linked as rules of thumb, assuming that the combination of links between the traces of two logs with the highest combined score for all these factors is probably better than other combinations. Therefore we defined the fitness function such that the fitness score is the sum of the individual scores for each factor we identified (see Formula 1):

$$f = w_1 \sum STI_i + w_2 \sum TO_i + w_3 \sum EAV_i + w_4 \sum NLT_i + w_5 \sum TD_i \,. \quad (1)$$

If more information about the relation between the two logs is known, weights could be configured by the user to reflect the relative importance of each factor (w_1-w_5). The following factors were taken into account when defining our fitness function:

Same trace id (STI_i)

A first factor to indicate that two traces belong together is if they have the same trace id (i.e. the process execution is consistently identified in both logs). In this case the problem is rather trivial, because it's almost certain how to merge the two log files. But this is no reason to exclude the factor from our fitness function. If, exceptionally, two traces with the same trace id do not belong together (e.g. a customer number that matches with an invoice number), then another solution should score higher due to the other factors of the fitness function.

Trace order (TO_i)

Because we assume all second traces are originated in the first log, the first event of the second trace should occur after the first event of the first trace (given our assumption regarding the reliability and comparability of event timestamps). All trace links that violate this rule make the fitness score decrease. We call this a punishment score and implement this simply by adding a negative score to the overall fitness score (i.e. $TO_i \leq 0$).

Equal attribute values (EAV_i)

In many processes a reference number or code is used throughout the entire process. This is most probably the trace id. But maybe other numbers are passed from event to event. If this number is logged, we should search for matching values of event attributes. Note that attribute names do not need to correspond. The name for this number can be different in both logs (e.g. "invoice number" and "reference number") and matching attribute names is more challenging [12]. Also note that some attribute values may have equivalents in lots of traces (for example status *completed*). This would make barely any difference between different solutions, because almost all possible solutions would score higher.

Number of linked traces (NLT_i)

Although multiple second traces can be linked to the same first trace, we consider this to be reasonably exceptional and therefore give a punishment score (i.e. $NLT_i \leq 0$) for each additional second trace linked to the same first trace. As with almost all other factors this is only an indication. If a lot of other factors indicate that multiple second traces belong to the same first trace, the fitness function must be defined such (e.g. by adapting the weights of the different factors) that their combined score is more positive than the (negative) punishment score for this factor. In this way it is still possible, but less probable, to end up with different second traces for the same first trace.

Time distance (TD_i)

This factor focuses on the actual distance in time between two traces (i.e. the difference between the time stamps of the first events of both traces). If two second traces are candidates to be linked to a same first trace and the fitness score for all other factors is equal for both solutions, we think the one with the least time distance to the first trace is more likely to be the correct matching partner. Because we have no idea about the correct time distance we did not score the actual distance, but we simply add to the score if time distance decreased (i.e. $TD_i > 0$) and subtract to the score if time distance increased (i.e. $TD_i < 0$).

3.2.3 Mutation and Termination

We can be short about the remaining properties of our implementation of the genetic inspired algorithm. In each step we select a certain fixed amount of second traces and

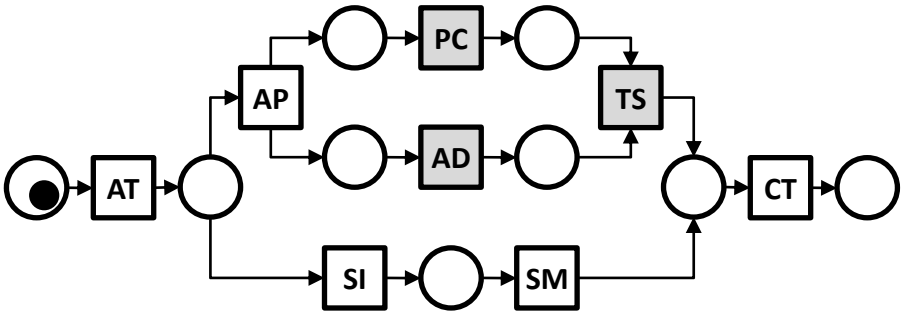
link them with another random first trace (this is a mutation). If the fitness function score at the end of this step is higher than before, we continue with this new link combination, otherwise we revert to the previous combination. As we are not (yet) searching for the fastest implementation of this algorithm we also use a fixed number of steps. There is no real termination condition. This can of course be changed later. When at a given point no progress is made in the last steps, it is perhaps better to stop the algorithm earlier.

4 Proof of Concept

We have tested our technique with a simulated example. The benefit of using simulation is that the correct solution (i.e. the process to be discovered) is known. Another advantage is that properties like time difference or noise can be controlled.

4.1 Example Model

Our example of a back-end IT support process (Fig. 3) is based on the same example model as in [5]. We generated two log files with 100 random executions of the process where the executions of tasks AT, AP, SI, SM and CT were logged in a first log file and the executions of tasks PC, AD and TS in the second file. We initially did not include noise, the executions did not overlap in time, and there was no structural unbalance in choosing one or the other path first for AND-split or selecting the path to be followed for an OR-split. Time differences between consecutive events were also random.



AT: Appoint Ticket, AP: Analyse Problem, PC: Program Changes, AD: Adjust Documentation
TS: Test Solution, SI: Search Information, SM: Send Mail, CT: Close Ticket

Fig. 3. Process model for our back-end IT support process example

4.2 Technical Implementation

We implemented our algorithm in the latest version of ProM [3] (ProM 6), a well-known academic process mining tool developed by the Process Mining Group at

Eindhoven University of Technology. This tool can be downloaded from http://prom.sf.net.

Genetic algorithms can be optimised by narrowing down the candidates in each random choice [6]. Instead of starting with a totally random initial linking combination, we start with a combination where each second trace is linked to a first trace with which it has the highest individual fitness score (i.e. fitness score for this particular link rather than for the entire solution of links between the traces of both logs). We also think some factors of the fitness function (like same trace id and trace order) are stronger than others. Therefore we used different weights for individual factors. Finally, because we want to improve our solution in every step, we do not select random traces for change in the mutation step. Instead we randomly select a factor and then select a random trace for which we can improve this factor.

4.3 Tests and Results

We tested log files for 100 sample model executions with our implemented log file merging algorithm using 100,000 steps. We included the following attributes in our test logs: name, originator, status, and time[2]. Because there is no unbalance in choosing paths in the OR-split, the second log ends up with about 50 traces. For a simple example of 100 generated traces with no noise, no overlapping traces, and matching trace ids the result is shown in Fig. 4 (after mining with Heuristic Miner [8]). A log-to-log comparison shows that all traces were correctly linked and merged between both files.

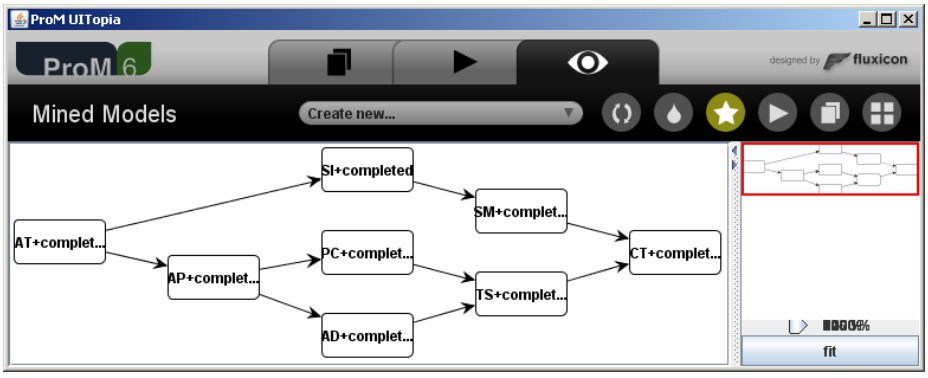


Fig. 4. Resulting model after merging and process discovery with Heuristic Miner [8]

In a second test run, we generated files for the same model with different properties:

[2] This corresponds to these extensions of XES ("an XML-based standard for event logs") used in ProM: concept:name (name), org:resource (originator), lifecycle:transition (status), and time:timestamp (time). (http://www.xes-standard.org/)

- We added noise in the same way as described in [8]. One of four options is randomly selected: (i) delete minimum one and up to one third of events from the beginning of a trace, (ii) from the middle of a trace, (iii) from the end of a trace, or (iv) switch places of two random events in a trace. The noise percentage determines the chance a trace is influenced by noise in one of the four ways. We tested with 0, 10, 20 and 50% noise.
- Another property we varied is overlap. The overlap percentage determines the chance of each execution to start during the previous execution. With 10% overlap 10% of traces started before the previous one ended. We did tests with 0, 10, 20, 50, 75 and 100% overlap.
- Finally, we repeated each test with a set of log files where trace numbers did never match.

The results of our tests can be derived from Table 1 (for matching id numbers) and Table 2 (for non-matching id numbers). When traces run partly in parallel (overlap > 0%) and no matching trace numbers exist, there seems to be too little information left to find a proper solution.

Table 1. Test results for matching ids with varying noise and overlap percentage (incorrect links in relation to total links identified)

Matching id	Overlap						
Noise	0	10	20	50	75	100	Mean
0	1/59	1/55	0/50	0/54	0/50	3/45	2%
10	0/49	0/43	0/62	0/50	0/46	0/54	0%
20	0/48	0/48	0/49	0/53	0/45	0/38	0%
50	0/46	0/54	0/64	0/56	0/41	0/49	0%
Mean	0%	0%	0%	0%	0%	2%	0%

Table 2. Test results for non-matching ids with varying noise percentage, overlap percentage (incorrect links in relation to total links identified)

No matching id	Overlap						
Noise	0	10	20	50	75	100	Mean
0	0/51	7/50	4/42	12/48	19/56	36/46	27%
10	0/57	1/44	0/49	10/57	14/46	27/50	17%
20	0/59	2/51	8/50	7/46	13/53	23/50	18%
50	1/47	6/53	1/47	13/56	16/46	27/50	21%
Mean	1%	8%	7%	20%	31%	58%	21%

We also performed tests with other amounts of process executions and with other simple models. Due to a page restriction we do not describe any of these other tests in this paper, but the test outcomes were similar to the results in Table 1 and Table 2.

5 Discussion

In the previous sections we presented our technique. In this section we discuss (i) performance of our solution, (ii) the assumptions we made, and (iii) the factors we used in the fitness function.

Performance of our implementation depends on m x ($2n_1$ + $2n_2$) = 100,000 x (100+100+50+50) = 3 x 10^7 sets of instructions to be executed (where m is the number of steps and n_1 and n_2 are the number of traces in the log files). If we would use a technique that looks for all possible solutions for merging each second trace to a first trace, n_1^{n2} = 100^{50}=10^{100} options should be checked (permutation with repetition of 50 elements out of a set of 100). This relates to the number of options in our solution as 3 x 10^{87} years to 4 minutes (about the mean time our algorithm ran in our tests). Off course, there may still be better solutions, we only claim to have found an optimal solution with an *acceptable* performance (for our example case).

Process mining is based on a number of assumptions (see 2.1). We made two additional assumptions (see 2.2). First, we assume the first log contains all process start events and we know which log is first. We expect to be able to drop this assumption later on.

The second assumption we made is the presence (and reliability and comparability) of event timestamps. We use this for matching traces and we really need this for merging traces. The only way to find out in which order events happened is to look at some time information. We think data is logged for different reasons, but is mostly concerned about what, who and when. Therefore in our opinion in most cases time information is logged, so the assumption focuses on reliability and comparability. Whereas our assumption may seem narrower than the assumption of correct order in process discovery, we think to have widened the possibilities of process discovery because in our opinion time information is mostly available but getting all process data in one log file is more challenging.

We still need to complete research on which factors and weights to use. Maybe some factors should be omitted, maybe others should be added. For example one factor we are experimenting with is time difference. We now assume the difference between first events in first and second traces should be minimal, but we are not sure this assumption is totally correct. A better assumption could be that time difference between all first and second traces should be alike, whether it is a large difference or a small one. Mathematically this is translated into a minimisation of the standard deviation of all time differences of a combination.

We also relied on assumptions made in other process mining research like for instance the perspective on noise. In [8], for example, noise was incorporated as one of four options. One of these options is to “interchange two random chosen events” ([8], p13), but we are not convinced that this type of noise is common. We think an event could be logged too late, but it does not seem logical that another event would

be logged earlier instead. We plan on looking into this kind of assumptions to verify applicability.

6 Conclusion

In this paper we presented a possible solution to still be able to use process mining techniques if process data is recorded in multiple log files. By merging the log files into a new log file all existing process mining techniques could still be used. Our technique is inspired by genetic algorithms and uses random operations which are evaluated by a combined score (see 3.1). We presented a possible set of factors to build up this score (see 3.2.2) and implemented the technique in the ProM tool (see 4.2). As a minimal form of evaluation we tested the implementation with generated files for a given model. We claim to have shown for at least one example case that this technique works and the test results are quite good (see 4.3). Lots of research has to be done to properly validate the technique and its implementation. We tried to give an overview and extended discussion of which elements need further research (see Section 5).

7 References

1. Rozinat, A., Mans, R.S., Song, M., Van Der Aalst, W.M.P. Discovering Simulation Models. Information Systems. 34, 305–327 (2009)

2. Van Der Aalst, W.M.P., Van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M. Workflow Mining: A Survey of Issues and Approaches. Data & Knowledge Engineering. 47, 237–267 (2003)

3. Van Dongen, B.F., De Medeiros, A.K.A., Verbeek, H.M.W., Weijters, A.J.M.M., Van Der Aalst, W.M.P. The ProM Framework: A New Era in Process Mining Tool Support. Applications and Theory of Petri Nets. p. 444–454 Springer (2005)

4. Van Der Aalst, W.M.P. Challenges in Business Process Analysis. Enterprise Information Systems. p. 27 Springer verlag (2008)

5. Van Der Aalst, W.M.P., Weijters, A.J.M.M. Process Mining: A Research Agenda. Computers in Industry. 53, 231–244 (2004)

6. Goldberg, D.E. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-wesley (1989)

7. Li, J., Liu, D., Yang, B. Process Mining: Extending α-Algorithm to Mine Duplicate Tasks in Process Logs. Advances in Web and Network Technologies, and Information management. pp. 396-407 Springer-Verlag New York Inc (2007)

8. Weijters, A.J.M.M., Van Der Aalst, W.M.P. Rediscovering Workflow Models from Event-based Data Using Little Thumb. Integrated Computer-Aided Engineering. 10, 151–162 (2003)

9. Weijters, A.J.M.M., Van Der Aalst, W.M.P. Process Mining: Discovering Workflow Models from Event-based Data. Belgium-Netherlands Conference on Artificial Intelligence. p. 283–290 Citeseer (2001)

10. Darwin, C. On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life. Murray, John, London (1869)

11. Ross, P., Hart, E., Corne, D. Genetic algorithms and timetabling. Advances in Evolutionary Computing. pp. 755-771 Springer-Verlag New York, Inc. (2003)

12. Wang, J.R., Madnick, S.E. The inter-database instance identification problem in integrating autonomous systems. Data Engineering. p. 46–55 IEEE (2002)